DOCUMENT RESUME

ED 449 178 TM 032 258

AUTHOR Brooks, Gordon P.; Barcikowski, Robert S.; Robey, Randall R.
TITLE Monte Carlo Simulation for Perusal and Practice.
PUB DATE 1999-04-00
NOTE 31p.; Paper presented at the Annual Meeting of the American Educational Research Association (Montreal, Quebec, Canada, April 19-23, 1999).
PUB TYPE Reports - Descriptive (141) -- Speeches/Meeting Papers (150)
EDRS PRICE MF01/PC02 Plus Postage.
DESCRIPTORS *Computer Simulation; *Monte Carlo Methods; *Research Methodology; Sampling
IDENTIFIERS Type I Errors; Type II Errors

ABSTRACT

The meaningful investigation of many problems in statistics can be solved through Monte Carlo methods. Monte Carlo studies can help solve problems that are mathematically intractable through the analysis of random samples from populations whose characteristics are known to the researcher. Using Monte Carlo simulation, the values of a statistic are observed in many samples drawn from that known population. The statistic's sampling distribution is then estimated by the relative frequency distribution actually observed in the study. The many samples are usually generated artificially through the use of computer algorithms. This paper discusses how one chooses a pseudo-random number generator, and then discusses how to use these generators to simulate data from normal and multivariate normal distributions. Several examples are provided of how one might sample from a pseudo population and the statistics that are commonly calculated during a Monte Carlo investigation. The many trials that might be used when investigating Type I and Type II errors are outlined. The paper ends with a discussion of the processes that might be used to verify that a researcher's simulation process is doing what it was intended to do. Four appendixes contain Statistical Analysis System programming code examples and a FORTRAN code example for generating data. (Contains 1 table and 38 references.) (SLD)

Reproductions supplied by EDRS are the best that can be made
from the original document.

Monte Carlo Simulation For Perusal and Practice

Gordon P. Brooks
Robert S. Barcikowski
Ohio University

Randall R. Robey
University of Virginia

ED 449 178

G. Brooks

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
This document has been reproduced as received from the person or organization originating it.
Minor changes have been made to improve reproduction quality.
Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

TM032258

Paper presented at the meeting of the American Educational Research Association, Montreal, Quebec, Canada (1999, April)

BEST COPY AVAILABLE

MONTE CARLO SIMULATION FOR PERUSAL AND PRACTICE

Ideally, a theoretical mathematical analysis would be offered to describe the efficiency of a statistical method (Halperin, 1976; Harwell, 1990). However, several conditions may not permit such an analytic analysis. For example, mathematical analysis is not possible when (a) statistical assumptions do not hold, (b) conditions required for mathematical theory are not met (e.g., the null hypothesis is known not to be true), or (c) the mathematics of the sampling distribution have not yet been worked out for a statistic (Mooney, 1997).

Monte Carlo Simulation

Fortunately, meaningful investigation of many problems in statistics can be solved through Monte Carlo methods. As noted by Mooney (1997), Monte Carlo simulation "offers an alternative to analytical mathematics for understanding a statistic's sampling distribution and evaluating its behavior in random samples" (p. 2). That is, a Monte Carlo study can help solve problems that are mathematically intractable.

Monte Carlo simulations perform these functions empirically through the analysis of random samples from populations whose characteristics are known to the researcher. Using Monte Carlo simulation, the values of a statistic are observed in many samples drawn from that known population. The statistic's sampling distribution is then estimated by the relative frequency distribution actually observed in the study. The many samples are usually generated artificially through the use of computer algorithms.

Monte Carlo methods use computer assisted simulations to provide evidence for problems that cannot be solved mathematically, such as when the sampling distribution is unknown or the null hypothesis is not true. Monte Carlo simulation often is used to study the robustness of a statistic (Kleijnen, 1974). That is, Monte Carlo conditions are set up to violate underlying assumptions of a specific statistic or procedure to determine how sensitive it is to the given violations. Kleijnen also described Monte Carlo studies designed to study the sampling distribution of a statistic or to compare the distributions of more than one estimate of a parameter.

For example, a researcher may develop a modified statistical test or procedure that is expected to be more robust to some underlying statistical assumption (e.g., non-normality or heterogeneity of variances). The researcher may use simulation to determine Type I error rates of the modification over many samples, which are created under a true null hypothesis but with violations of the assumption in question. The researcher would assess the actual proportion of Type I error levels, π, of the statistic, knowing that any rejection (i.e., statistically significant result) would be a Type I error under the true null hypothesis. Based on these results, the researcher may determine that the new modification is nearly equal ($\pi=\alpha$), more conservative ($\pi<\alpha$), or more liberal or inflated ($\pi>\alpha$) than the error levels chosen as acceptable by the researcher. For example, if the actual proportion of Type I error rates were $\pi=.10$, and therefore too far above the expected $\alpha=.05$, the new modification or procedure may not prove worthy of further consideration.

Statistical power analyses follow similar procedures, but generating data such that the null hypothesis is known to be false. The number of true rejections of the null hypothesis are counted for the series of simulated experiments in order to calculate empirical statistical power rates. After all samples are completed, a proportion is calculated that represents the actual

statistical power rate. As described for the Type I error situation, this rate may indicate that Type II errors are higher or lower than expected (recall that power is 1-β, where β is the probability of a Type II error).

Design of Monte Carlo Simulation Studies

Mooney (1997) has described the following steps for a basic Monte Carlo simulation:

1. Specify the pseudo-population in symbolic terms in such a way that it can be used to generate samples. This usually means developing a computer algorithm to generate data in a specified manner.
2. Sample from the pseudo-population (a *pseudo-sample*) in ways reflective of the statistical situation of interest, for example, with the same sampling strategy, sample size, and so forth.
3. Calculate θ in the pseudo-sample and store it in a vector, θ.
4. Repeat Steps 2 and 3 *n* times, where *n* is the number of *trials*.
5. Construct a relative frequency distribution of the resulting θ_i, values, which is the Monte Carlo estimate of the sampling distribution of θ under the conditions specified by the pseudo-population and the sampling procedures. (p. 4)

It should be noted that the Monte Carlo design is not very different from more standard research design, which typically includes identification of the population, description of the sampling plan, data collection, and data analysis. Although most of the Monte Carlo design is straightforward, Mooney (1997) noted that the two most difficult facets are writing the computer code to simulate the desired data conditions and interpreting the estimated sampling distribution. In what follows we consider in more detail each of Mooney's first four steps.

Step 1: Specify The Pseudo-Population

In this section we describe how random numbers are created on computers and how they are used to generate data from normal and multivariate normal distributions. We begin with a discussion of research that has been done on the center piece of random number generation, generating random number from a uniform distribution.

Uniform Number Generation. The uniform distribution, often designated U(0,1), is the building block of all Monte Carlo simulation work. From the uniform distribution, variables with all other distribution functions are derived (Mooney, 1997). This is because the uniform distribution has a range from zero to one, which can define a range for random probabilities. These uniform random numbers, then, can be used as probabilities to be input into other distribution functions using the appropriate inverse transformations and acceptance-rejection methods (Mooney, 1997).

Although there are a variety of methods by which more nearly and theoretically random uniform numbers can obtained, most Monte Carlo research generates these numbers by applying a deterministic algebraic formula. It is interesting to note that Intel is expected to include a hardware solution for random number generation in future chip sets (Miller, 1999). This process will generate random numbers by measuring the random thermal noise. The benefit to a hardware solution is that "for a variety of geeky reasons, it's virtually impossible to generate truly random numbers using software alone" (Miller, 1999, p. 58).

For practical purposes, the numbers produced by deterministic methods are considered to behave as random numbers if they have passed rigorous testing for uniformity and mutual

independence (Kleijnen, 1974; L'Ecuyer, 1988; Mooney, 1997; Rubinstein, 1981). Because these numbers are generated in a deterministic fashion, they are said to be pseudorandom. The primary advantage of deterministic generators over the use of random numbers chosen in some other fashion is that a sequence of pseudorandom numbers can be reproduced with repeated use of the same initial seed value (Kleijnen, 1974; Mooney, 1997). The most important disadvantage is that after some period or cycle, the same sequence of numbers begins to repeat (Kleijnen, 1974).

Linear congruential generators (LCG) are the most commonly used generators for the uniform distribution (Bratley, Fox, & Schrage, 1987; Knuth, 1981; L'Ecuyer, 1988). Linear congruential generators follow the algebraic formula:

$$X_{i+1} = (aX_i + c) \text{ MOD } m,$$

where a is called the multiplier, c is called the increment, and m is called the modulus (Knuth, 1981). The modulo function, MOD, takes as it result the remainder of a division (e.g., 10MOD3=1). Also required is an initial seed value X_0 to serve as the starting value for the pseudorandom sequence. The modulus usually is chosen to be a large prime integer and the multiplier is some integer less than m (Park & Miller, 1988). When c = 0, a special class of linear congruential generator called multiplicative congruential generators result (Bratley, Fox, & Schrage, 1987; Kleijnen, 1974; Mooney, 1997). A multiplicative congruential generator (MCG) is more computationally efficient because the addition operation is not performed (Fishman & Moore, 1982).

Pseudorandom numbers generated through any procedure should be tested, especially for uniformity and independence (Kleijnen, 1974). For instance, for any LCG, the values chosen for a, c, m, and X_0, which Knuth called "magic numbers" (p. 9), determine the statistical quality of the generator and therefore are usually chosen for their proven performance. As stated by Bratley, Fox, and Schrage (1987), "linear congruential generators are not universally reliable" (p. 224). In particular, some LCGs produce numbers that cause the common Box-Muller method for generating random normal deviates to be very poor. For example, the use of the Box-Muller transformation in conjunction with a multiplicative congruential generator can result in too few large normal values (Hauck & Anderson, 1984) or data that falls along a spiral if graphed in two dimensions (Bratley, Fox, & Schrage, 1987).

Also, Knuth (1981) showed that one particularly poor combination of values for a, c, m, and X_0 produces the sequence: 7, 6, 9, 0, 7, 6, 9, 0,..., for which the period is only four. Several studies have compared values for a, c, and m constants (e.g., Fishman & Moore, 1982; L'Ecuyer, 1988; Park & Miller, 1988; Wichmann & Hill, 1982). For example, Park and Miller (1988) compared many combinations of these constants against what they called the minimal standard generator (a = 16807 and $m = 2^{31} - 1$), that is:

$$f(z) = 16807z \ \textbf{\textit{MOD}}\ 2147483647$$

which is chosen because (a) it is known to be full period (i.e., all numbers less than m are produced), (b) it is demonstrably random, (c) it can be implemented correctly on almost any computer system, and (d) it has been tested exhaustively and as a result its characteristics are well understood.

Park and Miller (1988) have suggested that a generator that combines individual generators (e.g., Wichmann & Hill, 1982) is a logical extension of their minimal standard generator. Indeed, Golder and Settle (1976) have recommended that when the Box-Muller algorithm for generating random normal deviates is used, two MCGs should be combined. In

particular, L'Ecuyer (1988) has described a method by which two MCGs can be combined efficiently to produce a generator that, on a 32-bit computer, has a period of approximately 8.1×10^{18}, compared to a period of about 10^8 for the minimal standard generator (Press, Teukolsky, Vetterling, & Flannery, 1992). The combined generator recommended for 32-bit computers by L'Ecuyer (1988) uses a=40014 and m=2147483563 for one generator and a=40692 and m=2147483399 for the second. Press, Teukolsky, Vetterling, and Flannery (1992) have implemented L'Ecuyer's (1988) generator with an additional shuffle, which is used to break up serial correlations in the sequence produced by a generator. They have wagered (literally) that this generator "provides perfect random numbers; a practical definition of 'perfect' is that we will pay $1000 to the first reader who convinces us otherwise" (p. 272).

Following the recommendation of Bratley, Fox, and Schrage (1987) that one should "seek protection by using a hybrid generator not known to have pathologies but having theoretical support" (p. 224), the L'Ecuyer (1988) generator is recommended. A combined generator is recommended for two reasons. First, for more extreme cases of the Monte Carlo simulation, the number of independent pseudorandom numbers may exceed the period of most generators. Press, Teukolsky, Vetterling, and Flannery (1992) have noted that the minimal standard generator is not known to fail any statistical test, "except when the numbers of calls starts to become on the order of the period m, say $>10^8$" (p. 271). Second, it has been recommended that a combined generator be used when the Box-Muller method is used to generate random normal deviates. Finally, based on Mooney's (1997) recommendation, the generator should be "reseeded" occasionally, e.g., at the beginning of each outer loop, to ensure no problems with periodicity.

Normal Deviate Generation. One of the most popular methods for generating normally distributed data is the Box-Muller transformation (Golder & Settle, 1976). The Box-Muller method, called the polar method for normal deviates by Knuth (1981), applies the transformations

$$z_1 = \sqrt{(-2ln u_1)}\,(\cos 2\pi u_2)$$

and

$$z_1 = \sqrt{(-2ln u_1)}\,(\sin 2\pi u_2)$$

to two independent uniform numbers, u_1 and u_2, to obtain two independent standard normal deviates, z_1 and z_2 (Golder & Settle, 1976; Knuth, 1981; Press, Flannery, Teukolsky, & Vetterling, 1989; Rubinstein, 1981). Golder and Settle (1976) have argued that no other method for generating normal data has the simplicity and computational efficiency of the Box-Muller transformation; Bratley, Fox, and Schrage (1987) have characterized it as an "ingenious method" (p. 161).

Joint Multivariate Normal Data Generation. The correlation matrices that may be created to generate multivariate normal data following a Cholesky decomposition procedure (also known as the square root method) recommended by several scholars (Bratley, Fox, & Schrage, 1987; Chambers, 1977; International Mathematical and Statistical Library, 1985; Karian & Dudewicz, 1991; Kennedy & Gentle, 1980; Knuth, 1981; Mooney, 1997; Morgan, 1984; Ripley, 1987; Rubinstein, 1981). The Cholesky decomposition of a matrix produces a lower triangular matrix, L, such that

$$\mathbf{L}\mathbf{L}^{\mathbf{t}} = \Sigma$$

where Σ is a symmetric, positive definite matrix such as a covariance or correlation matrix. This

lower triangular matrix, ***L***, can be used to create multivariate pseudorandom normal variates through the equation

$$\mathbf{Z} = \mu + \mathbf{XL}^{T}$$

where ***Z*** is the multivariate normal data matrix containing Z_{ij}, μ is the vector containing the variable means μ_j, and ***X*** contains vectors of independent, standard normal variates. When μ=0, the multivariate pseudorandom data are distributed with mean vector zero and covariance matrix Σ.

Step 2: Sample From The Pseudo-Population

In this section four methods for generating data from a pseudo-population will be discussed. The first three of these methods will be described and illustrated using SAS programming commands and the last using FORTRAN commands. For didactic purposes we will investigate what happens to the actual Type I error rate of the single group t-test when its' assumption of normality is violated. For the first three methods we will violate the assumption of normality by using the *contaminated normal distribution* discussed by Andrews, D. F., Bickel, P. J., Hampel, F. R., Huber, P. J., Rogers, W. H. & Tukey, J. W. (1972). For the fourth method we will use data described by Sawilowsky and Blaire (1992).

We created a contaminated normal distribution by generating a proportion of the scores from a normal distribution with a mean of zero and a standard deviation of one, *i.e.*, a *standard normal distribution*, and included among these scores a proportion of the scores from a normal distribution with a mean of zero and a standard deviation of four. These latter scores were the contaminates. This was done for four situations where the proportions of scores from the standard normal distribution were: 1.00 (no violation), .90, .80, and .60, and therefore, the proportions of contaminates were: 0, .10, .20, and .40. For each of these situations, we performed a single group one-tailed t-test on a sample of ten units. We then randomly generated samples of ten units and calculated a single group one-tailed t-test 1,000 times for each of the four situations. For purposes of illustration we also repeated this process 10,000; 50,000 and 100,000 times. The latter numbers are referred to as the number of replications or iterations and the authors of most Monte Carlo studies only choose one such number.

The question of interest using the first three methods in this study was: Will there be differences among the four contaminating situations with respect to the one-tailed Type I error of the single group t-test? In this study the Type I error rate (referred to as the *nominal* level of significance) was set at .05. We kept track of the *actual* level of significance in each contaminating situation by adding up the number of times we rejected the null hypothesis that the group mean was equal to zero. For example, if when we generated the 1,000 replications we counted 39 rejections when the proportion of contaminants was .20, then our actual alpha would be found to be 39/1000 = .039. When this result is compared with the number of rejections in the no violation situation, approximately .05, we would say that our test was performing conservatively under this violation of normality.

(1) Standard Method: Generate Data Using Different Random Number Seeds. As was described at Step 1, in most Monte Carlo studies the data are generated by computer using what is called a *pseudo-random number generator*. The process is begun by supplying the generator with a *seed* value which is a number usually of between 5 to 9 randomly selected digits. [These seed values may be selected from tables of random numbers found in most statistics books. Or, you may select one seed value from a table of random numbers and use this seed to have your

computer generate further seed values.] In most studies a different seed value initiates each violation situation. This process is illustrated in the SAS program found in Appendix A. In this program the four data contamination situations were created in four sections of the program: (1) DATA NORMAL, (2) DATA NORM_10, (3) DATA NORM_20, and (4) DATA NORM_40. Following the CARDS statement in each of these sections are three or four numbers. The first number is the seed value, the second number is the size of each sample (10), the third number (here, 100,000) is the number of replications, and the fourth number is the proportion of contaminants. The first situation has no contaminants and has no fourth number. The respective seed values in these sections are: 3445167, 9221023, 5787747, and 1272301.

The estimates of Type I error from each data creation section and for each number of replications (ITERATE) are shown following the SAS program code. In this output, the different contaminating situations are labeled OBS and numbered 1 (no contamination), 2 (.10), 3 (.20), and 4 (.40). The seed values are in the column labeled SEED, the sample size is in the column labeled SAMPLE, the number of rejections are in the column labeled CK and the actual Type I error rate is shown under the heading ACTUAL. The differences among the actual Type I error rates are shown in the line under the latter values beginning with the labels OBS and DIF_12. For example, given 1,000 replications, the difference between the Type I error rate of .046 for the no contamination situation and .044 for the .10 contamination situation is .002. You may observe that as the number of replications increases to 100,000 the differences among the actual Type I error rates begin to stabilize.

(2) Olson's Method: Generate A Single Set Of Data And Perform All Treatments On These Data And Use Monte Carlo Critical Values. In his dissertation, Olson (1973) generated a single set of random data and then performed different treatments on these data. In our context this is equivalent to using the same set of random numbers in each situation and then contaminating some (a fixed proportion) of them. Olson did this..."so that comparisons among factor combinations would have much higher precision than if they were diluted by fluctuations between batches of random numbers." (p. 35).

Olson also used what he called a *Monte Carlo critical value*. If ITERATE is the number of replications, and if we arrange the t values from our non-contaminated group in order from largest to smallest, then the mean of (ITERATE times .05)th and of (ITERATE times .05 + 1)th largest of the ITERATE values is the Monte Carlo critical value. For example in our data when ITERATE = 1,000, the Monte Carlo critical value was the average (1.77145) of the 50^{th} = 1.76615 and 51^{th} = 1.77676 largest values.

The SAS program illustrating Olsen's method is shown in Appendix B, and the output from this program follows. The same set of random numbers can be generated each time by using the same seed to initiate the sequence. The seed used here was 3445167, and the Monte Carlo critical values (MCCV) were ITERATE = 10,000, MCCV = 1.77012; ITERATE = 50,000, MCCV = 1.81086; ITERATE = 100,000, MCCV = 1.82323. The theoretically derived tabled t value (one-tailed, α = .05, df = 9) is 1.833. Note that because of the method of finding the Monte Carlo critical value the actual level of significance in the output is .05 in the non-contaminated situation across all sizes of ITERATION.

(3) Modified Olson Method. Another approach to generating Monte Carlo data is a modification of Olson's method where instead of using Monte Carlo critical values one uses theoretically derived t-values. The results from this approach using the SAS program reported in Appendix A with the constant seed value of 3445167 are presented in Appendix C. These results

compare favorable to those presented in Appendix B.

(4) Create A Population Like Those Found In Real Life And Take Random Samples From This Population. Micceri (1989) investigated the characteristics of 440 large-sample achievement and psychometric measures and found the distributions of all of these measures to be nonnormal. Sawilowsky and Blair (1992) investigated the Type I and II error properties of the t-test by generating data from populations like those found by Micceri. Appendix D contains the Fortran code used by Sawilowsky and Blair to generate random samples of discrete data from what Micceri called a multimodal lumpy achievement score distribution. The output for 1,000 iterations follows the program code.

Step 3: Calculate θ In The Pseudo-Sample

Some common statistics calculated during a Monte Carlo simulation study include Type I errors, statistical power, bias, root mean square error (or deviation), and relative efficiency. More formal definitions are provided below.

Type I Error. By setting the parameters for the study the researcher is able to manipulate the conditions such that, as was shown in the examples for Step 2, the statistical null hypothesis is known to be true in each sample. By forcing the population to reflect a true null hypothesis, a researcher performs a Monte Carlo study to test the robustness of a statistic, usually to violations of assumptions. Type I error is calculated as the proportion of the total number of false rejections of the true null hypothesis to the total Monte Carlo samples performed. For example, if 10,000 Monte Carlo samples are created and the nominal level of significance is set at α=.05, if 800 samples are actually found to be statistically significant (a rate of θ=.08), then the Type I error rate is considered to be liberal because more than the expected 500 samples were significant. Fewer significant samples than expected would result in a conclusion that the statistic is conservative, and hence robust to the particular violation of the assumptions.

Statistical Power. Statistical power will be calculated as the proportion of the total number of correct rejections to the total tests performed for each testing situation. Because the statistical null hypothesis is known to be false in each sample of a statistical power study, each rejection at an α=.05 significance level (for example) qualifies as a correct rejection and will be recorded as such. For each of these conditions, then, empirical statistical power rates will be calculated simply as the proportion of the tests for each sampling condition that were correctly rejected.

Bias. Because the relative frequency distribution of the simulated θ_i estimates the sampling distribution of θ, the characteristics of the frequency distribution can be informative (Mooney, 1997). For example, the central tendency can be used to estimate a statistic's bias; the variability of a statistic can be used to compare two statistics in terms of efficiency (Mooney, 1997).

Statistical bias is defined as the difference between the population value and the expected value of its estimate in the sample (Drasgow, Dorans, & Tucker, 1979; Kromrey & Hines, 1995; Mooney, 1997). Specifically, the central tendency of the sampling distribution for a statistic allows us to estimate that statistic's bias:

$$Bias = E(\theta) - \theta,$$

where θ is the population parameter and $E(\theta)$ is the expected value of the sample statistic (average of the statistic over infinite samples).

In a Monte Carlo simulation, unlike real life, the population parameter is a known value

that is set to a given value in the computer algorithm. Therefore, bias is estimated within a Monte Carlo study through the calculation of the simple difference between the mean of the parameter estimate over all samples created for a given condition and the known population parameter for that condition:

$$Bias = (\Sigma\theta_i)/n - \theta,$$

where θ is the known population parameter (as set in the computer algorithm), θ_i is the estimate of that parameter obtained in sample *i* of the Monte Carlo simulation, and *n* is the total number of samples taken in the Monte Carlo study.

RMSE. The Root Mean Squared Error (*RMSE*) is another common statistic used to evaluate the results of Monte Carlo simulations. *RMSE* provides an indication of the statistic's variability. Unlike the statistical bias, which is calculated after all the simulations have been performed, mean squared error is the average of the squared differences between the population parameter and its estimate for each sample. *RMSE*, then, is the square root of the mean squared error for the given statistic. That is,

$$RMSE(\theta) = \sqrt{\Sigma(\theta - \theta_i)2} \ / \ \sqrt{n},$$

where θ is the known population parameter (as set in the computer algorithm), θ_i is the estimate of that parameter obtained in sample *i* of the Monte Carlo simulation, and *n* is the total number of samples taken in the Monte Carlo study (Darlington, 1996; Drasgow, Dorans, & Tucker, 1979; Kennedy, 1988; Mooney, 1997)

Relative Efficiency. Mooney (1997) defined relative efficiency as the ratio of two *RMSE* values, multiplied by 100 to convert it to a percentage:

$$Relative\ Efficiency = 100 \times RMSE(\theta_A)/RMSE(\theta_B),$$

where θ_A and θ_B are two different estimates the same parameter (Mooney, 1997). That is, in order to compare to two estimates, θ_A and θ_B, of a particular population parameter, the researcher converts their respective *RMSE* values into a relative efficiency percentage. Values under 100 would indicate the superiority of estimator θ_A (i.e., θ_A with smaller *RMSE*), whereas values over 100 would represent the superiority of θ_B. Wichern and Churchill (1978) have used a similar efficiency criterion in order to compare two estimators directly.

Step 4: Repeat Steps 2 And 3 *n* Times

How many times should I estimate θ? In this section we discuss how a researcher might select the number of trials for their simulation study when θ estimates Type I or Type II error.

Assessing the error properties of an algorithm (i.e., α or Type I error, and β or Type II error) requires that the researcher not only define criterion values (e.g., α=.01 or β=.01), but also tolerances for departures form those values. Since it is unreasonable for an observed error rate obtained through simulation to equal exactly the nominal level, the researcher must set an interval about the nominal error rate (either α or β) that defines the limits of tolerable departure. Said differently, a researcher must not only define a nominal error level of interest (e.g., $\alpha = 0.05$), but also define what constitutes a meaningful departure from that nominal level.

By setting the unilateral width of an interval about α as some fraction of α, Bradley (1978) defined tolerance intervals for departures from nominal α in studies of robustness. Bradley (1978, p. 146) defined a 'fairly stringent criterion' as $\alpha \pm 1/10\alpha$, and a 'liberal criterion' as $\alpha \pm 1/2\alpha$. In this paper, these criteria are supplemented by an *intermediate criterion* (i.e.

$\alpha \pm 1/4\alpha$) and a *very liberal criterion* (i.e. $\alpha \pm 3/4\alpha$) in an effort to address a range of interests. In a more general sense, Bradley's definitions/intervals can be considered as definitions/intervals for departures for β as well. That is, Bradley's intervals can be generalized from definitions/intervals regarding tolerance for acceptable Type I error to definitions/intervals regarding tolerance for either kind of error (i.e., Type I or Type II). As such, we can think of the four intervals as levels of error tolerance (gradations from conservative to liberal). Within this framework, the application of hypothesis te. ting logic is straightforward and can yield the number of iterations necessary to detect meaningful departures from the null hypothesis under tolerable error probabilities (Robey & Barcikowski, 1992).

The general form of the non-directional null hypothesis that a population proportion (i.e., π which is estimated as $\hat{\pi}$ through simulation) equals some constant (c) is written as $H_0 : \pi = c$. In Monte Carlo experiments, this null hypothesis is adapted to the form of $H_0 : \alpha = c$ in studies of robustness (where α represents Type I error of the algorithm under test) or as $H_0 : 1 - \beta = c$ in studies of statistical power (where β represents Type II error of the algorithm under test). The tenability of null hypotheses like these can be directly evaluated through applications of the two-tailed proportions test.

Since the statistical power of the two-tailed proportions test to detect departures of π from c is not the same when $\pi > c$ as it is when $\pi < c$, comparison of π and c are facilitated by transforming each using

$$\phi_p = 2 \arcsin \sqrt{p}$$

where arcsin is given in radians and p is a proportion (Cohen, 1988). The value of $|\phi_c - \phi_\pi|$ is tested against the critical value

$$Z_{1-\omega/2} \sqrt{2/n} \sqrt{.5}$$

where Z is the standard unit normal deviate, and where ω is the Type I error rate of the proportions test (Cohen, 1988, pp. 548 and 212).

Once the Type I error rate (ω) and the statistical power level ($1 - \gamma$) for the proportions test have been selected, the necessary number of iterations (n) is given by Cohen (1988, p.549) as

$$n = 2 \left[\frac{Z_{1-\omega/2} + Z_{1-\gamma}}{|\phi_{c'} - \phi_\pi| \sqrt{2}} \right]^2$$

where c' is the upper bound (i.e., greater value) of the error-tolerance interval. The upper bound is used in the calculation of n since the arcsin transformation causes the interval about ϕ_c to be asymmetric where the distance from ϕ_c to $\phi_{c'}$ is less than the distance from ϕ_π to the transformed lower bound. As a result, critical departures from c in the direction of .5 are harder to detect than critical departures from c that are further out in the tail. The former departures, therefore, require a few more observations than do the latter to achieve criterion power. When an investigator is not interested in detecting conservative departures from c (i.e., values less than

c), n is accordingly calculated for a one-tailed test.

Computing resources can be conserved when examining multiple levels of c by testing the null hypothesis at the lowest interesting value of c only. Using the sample size necessary to detect departures from the lowest value of c in an effort to detect departures from greater c values would result in a very high level of statistical power in the application of the proportions test. Further, no advantage would be realized by analyzing the greater values of c using lesser values of n. As a result, the $\hat{\pi}$ values for the greater values of c might just as well be based on the greatest n and interpreted as population values.

Tabled values of n. Table 1 contains the number of iterations necessary for the two-tailed proportions test to detect departures from c relative to each of the four error-tolerance definitions. In this table, the values for c include .10, .05, and .01. For each c, the entries are indexed by three Type I error rates for the two-tailed proportions test, ω (i.e. ω = .01, .05, and .10), and by three statistical powers for the two-tailed proportions test, $1-\gamma$ (i.e., $1-\gamma$ = .7, .8, and .9).

From a hypothesis-testing-logic perspective, Table 1 reveals that the frequently encountered practice of carrying out 1000 iterations is appropriate only for larger values of c in combination with more liberal definitions of error tolerance. The detection of a departure from a small value of c, say .01 or less, from one of the more stringent error-tolerance definitions requires an n which is substantially larger than that found in many Monte Carlo studies. For example, 121312 iterations are required to detect departures from the most stringent definition of error tolerance at c=.01, when the power for the proportions test is set at .80 and ω is set at .01.

Verification of the Data Collection Procedures

According to Bratley, Fox, and Schrage (1987), verification of the algorithms should include (a) manual verification of the logic by comparing results of the computer analysis with results calculated by hand, (b) modular testing to ensure that each subroutine produces sensible output for all possible inputs, (c) checking the results against known solutions, (d) sensitivity testing to ensure that the behavior of the computer model is sensible when parameters are varied, and (e) stress testing to ensure that strange values do not cause unexpected problems. Each of these steps was performed in preliminary analyses to verify program integrity. As changes in the program occurred as it developed, testing was repeated.

Manual verification of the logic of the computer code can be performed in several ways. Most of these verifications require that the program be run for either a single sample or for a few samples so that hand calculations and counts can be made easily. First, several of the variables may be derived mathematically from other of the statistical information gathered for each sample, and therefore can be calculated for single or several samples by hand to compare to program results. Similarly, the *RMSE* for several statistics can be verified to produce expected results based on the values of the statistics. Also, the number of rejections of the statistical hypothesis are counted to ensure the correct number for whatever type of study is being performed. Manual verification also entails verifying the aggregated output over several iterations. Specifically, to ensure that averages taken over the total number of iterations are correct, output is produced for several samples and averages are calculated by hand (well, by hand with the aid of a calculator). Similar procedures can be performed for the *RMSE*, standard deviations, and number of rejections compiled.

Modular testing is performed in a similar manner to manual verification. For example,

the adaptations of the pseudorandom number generators used in a computer program can be tested to ensure that they do indeed produce uniformly and normally distributed data, as required. The Cholesky decomposition procedure, or some other procedure to generate multivariate normal data, must be tested with several matrices to ensure its performance. Data from the pseudorandom number generators should be verified to have appropriate means and variances. Finally, algorithms adapted from other sources can be verified to produce the same output as their original sources. The results of the statistical modules usually are verified by comparison to results from an accepted statistical package, such as SAS or SPSS.

Summary

In this paper we have discussed and illustrated the steps an educational researcher would take to complete a Monte Carlo study. We began by discussing how one chooses a pseudo-random number generator and then we discussed how to use these generators to simulate data from normal and multivariate normal distributions. We then provided several examples of how one might sample from a pseudo-population; what statistics are commonly calculated during a Monte Carlo investigation, and how many trials might be used when investigating Type I and Type II error. We completed the paper with a discussion of the processes one might use to verify that a researcher's simulation process is indeed doing what it was intended to do.

References

Andrews, D. F., Bickel, P. J., Hampel, F. R., Huber, P. J., Rogers, W. H. & Tukey, J. W., (1972). Robust estimates of location: Survey and advances. Princeton, NJ: Princeton University Press.

Borland International, Inc. (1990). Turbo Pascal (Version 6.0) [Computer Software]. Scotts Valley, CA: Authors.

Bradley, J. V. (1978). Robustness? British Journal of Mathematical and Statistical Psychology, 31, 144-152.

Bratley, P., Fox, B. L., & Schrage, L. E. (1987). A guide to simulation (2nd ed.). New York: Springer-Verlag.

Chambers, J. M. (1977). Computational methods for data analysis. New York: John Wiley & Sons.

Cohen, J. (1988). Statistical Power Analysis for the Behavioral Sciences, 2nd ed. Hillsdale, NJ: Erlbaum.

Darlington, R. B. (1968). Multiple regression in psychological research and practice. Psychological Bulletin, 69, 161-182.

Darlington, R. B. (1996). Estimating the true accuracy of regression predictions. Mid-Western Educational Researcher, 9(4), 29-31.

Drasgow, F., Dorans, N. J., & Tucker, L. R. (1979). Estimators of the squared cross-validity coefficient: A Monte Carlo investigation. Applied Psychological Measurement, 3, 387-399.

Fishman, G. S., & Moore, L. R. (1982). A statistical evaluation of multiplicative congruential random number generators with modulus 2^{31}-1. Journal of the American Statistical Association, 77, 129-136.

Golder, E. R., & Settle, J. G. (1976). The Box-Muller method for generating pseudo-random normal deviates. Applied Statistics, 1, 12-20.

Halperin, S. (1976, April). Design of Monte Carlo studies. Paper presented at the meeting of the American Educational Research Association, San Francisco, CA. (ERIC Document Reproduction Service No. ED 121 850)

Harwell, M. R. (1990, April). Summarizing Monte Carlo results in methodological research. Paper presented at the meeting of the American Educational Research Association, Boston, MA. (ERIC Document Reproduction Service No. ED 319 775)

Hauck, W. W., & Anderson, S. (1984). A survey regarding the reporting of simulation studies. American Statistician, 38, 214-216.

Hoaglin, D. C., & Andrews, D. F. (1975). The reporting of computation-based results in statistics. American Statistician, 29, 122-126.

International Mathematical and Statistical Library. (1985). Stat/PC Library. Houston, TX: Authors.

Karian, Z. A., & Dudewicz, E. J. (1991). Modern statistical systems, and GPSS simulation: The first course. New York: Computer Science Press.

Kennedy, E. (1988). Estimation of the squared cross-validity coefficient in the context of best subset regression. Applied Psychological Measurement, 12, 231-237.

Kennedy, W. J., Jr., & Gentle, J. E. (1980). Statistical computing. New York: Marcel Dekker.

Kleijnen, J. P. C. (1974). Statistical techniques in simulation (Part I). New York: Marcel

Dekker.

Knuth, D. E. (1981). The art of computer programming: Vol. 2. Seminumerical algorithms (2nd ed.). Reading, MA: Addison-Wesley.

Kromrey, J. D., & Hines, C. V. (1995). Use of empirical estimates of shrinkage in multiple regression: A caution. Educational and Psychological Measurement, 55, 901-925.

L'Ecuyer, P. (1988). Efficient and portable combined random number generators. Communications of the ACM, 31, 742-749, 774.

Micceri, T. (1989). The unicorn, the normal curve, and other improbable creatures. Psychological Bulletin, 105 (1), 156-166.

Mooney, C. Z. (1997). Monte Carlo simulation (Sage University Paper series on Quantitative Applications in the Social Sciences, series no. 07-116). Thousand Oaks, CA: Sage.

Morgan, B. J. T. (1984). Elements of simulation. New York: Chapman and Hall.

Nash, J. C. (1990). Compact numerical methods for computers: Linear algebra and function minimisation (2nd ed.). New York: Adam Hilger.

Olson, C. L. (1972). A monte carlo investigation of the robustness of multivariate analysis of variance. Unpublished doctoral dissertation, University of Toronto, Toronto, Canada.

Park, S. K., & Miller, K. W. (1988). Random number generators: Good ones are hard to find. Communications of the ACM, 31, 1192-1201.

Press, W. H., Flannery, B. P., Teukolsky, S. A., & Vetterling, W. T. (1989). Numerical recipes in Pascal: The art of scientific computing. New York: Cambridge University.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). Numerical recipes in FORTRAN: The art of scientific computing (2nd ed.). New York: Cambridge University.

Ripley, B. D. (1987). Stochastic simulation. New York: John Wiley & Sons.

Robey, R. R., & Barcikowski, R. S. (1992). Type I error and the number of iterations in Monte Carlo studies of robustness. British Journal of Mathematical and Statistical Psychology, 45, 283-288.

Rubinstein, R. Y. (1981). Simulation and the Monte Carlo method. New York: John Wiley & Sons.

Sawilowsky, S. S., & Blair, R. C. (1992). A more realistic look at the robustness and type II error properties of the t test to departures from population normality. Psychological Bulletin, 111 (2), 352-360.

Spath, H. (1992). Mathematical algorithms for linear regression. Boston: Academic.

Wichern, D. W., & Churchill, G. A. (1978). A comparison of ridge estimators. Technometrics, 20, 301-311.

Wichmann, B. A., & Hill, I. D. (1982). An efficient and portable pseudo-random number generator. Applied Statistics, 38, 188-192.

Table 1
Necessary Iterations For Two-Tailed Proportions Test To Detect Departures Of π From α.

c	$1-\gamma$	ω	$c\pm1/10c$	$c\pm1/4c$	$c\pm1/2c$	$c\pm3/4c$
		.10	4419	750	204	94
	.7	.05	5796	983	268	128
		.01	9027	1531	417	200
		.10	5810	986	269	129
.10	.8	.05	7375	1251	341	163
		.01	10973	1861	507	243
		.10	8047	1365	372	178
	.9	.05	9873	1675	456	218
		.01	13980	2371	846	309
		.10	9356	1594	437	211
	.7	.05	12271	2091	573	276
		.01	19111	3256	893	430
		.10	12301	2096	575	277
.05	.8	.05	15614	2660	729	351
		.01	23233	3958	1085	523
		.10	17038	2903	796	383
	.9	.05	20902	3561	976	470
		.01	29600	5042	1382	666
		.10	48852	8348	2300	1113
	.7	.05	64072	10948	3017	1460
		.01	99790	17051	4678	2274
		.10	64227	10975	3024	1464
.01	.8	.05	81527	13931	3838	1858
		.01	121312	20729	5711	2764
		.10	88963	15201	4188	2027
	.9	.05	109141	18649	5138	2487
		.01	154556	26409	7276	3521

Note: The value of c is a criterion proportion for either Type I error (α) or Type II error ($1-\beta$) of the algorithm being tested. The value of ω and $1-\gamma$ are *a priori* Type I error and statistical power levels of the two-tailed proportions test.

Appendix A

SAS Programming Code Illustrating
The Standard Method Of Generating Random Data
Using Different Random Number Seeds.

```
DATA NORMAL;
INPUT SEED SAMPLE ITERATE;
DO I = 1 TO ITERATE;
TOTX = 0;
SSX = 0;
N = 0;
  DO J = 1 TO SAMPLE;
  X = NORMAL(SEED);
  TOTX = TOTX + X;
  SSX = SSX + X * X;
  N+1;
  END;
XBAR = TOTX / N;
VARX = (SSX - N * XBAR**2) / (N-1);
T = XBAR / SQRT(VARX/N);
DF = N - 1;
P_T = 1-PROBT(T,DF);
   IF P_T < = .05 THEN DO;
   CK + 1;
   ACTUAL = CK / ITERATE;
   END;
KEEP SEED SAMPLE ITERATE CK ACTUAL;
END;
CARDS;
3445167  10  100000
;
DATA NORM_10;
INPUT SEED SAMPLE ITERATE PROP;
SAM1 = PROP * SAMPLE;
SAM2 = SAMPLE - SAM1;
DO I = 1 TO ITERATE;
TOTX = 0;
SSX = 0;
N = 0;
  DO J = 1 TO SAM2;
  X = NORMAL(SEED);
  TOTX = TOTX + X;
  SSX = SSX + X * X;
  N+1;
*PUT X= TOTX= SSX= N=  SAM2=;
  END;
  DO J = 1 TO SAM1;
  X = NORMAL(SEED) + 4;
  TOTX = TOTX + X;
  SSX = SSX + X * X;
  N+1;
*PUT X= TOTX= SSX= N=  SAM1=;
  END;
```

```
XBAR = TOTX / N;
VARX = (SSX - N * XBAR**2) / (N-1);
T = XBAR / SQRT(VARX/N);
DF = N - 1;
P_T = 1-PROBT(T,DF);
   IF P_T < = .05 THEN DO;
   CK + 1;
   ACTUAL = CK / ITERATE;
   END;
KEEP SEED SAMPLE ITERATE CK ACTUAL;
END;
CARDS;
9221023  10  100000  .10
;
DATA NORM_20;
INPUT SEED SAMPLE ITERATE PROP;
SAM1 = PROP * SAMPLE;
SAM2 = SAMPLE - SAM1;
DO I = 1 TO ITERATE;
TOTX = 0;
SSX = 0;
N = 0;
  DO J = 1 TO SAM2;
  X = NORMAL(SEED);
  TOTX = TOTX + X;
  SSX = SSX + X * X;
  N+1;
*PUT X= TOTX= SSX= N=  SAM2=;
  END;
  DO J = 1 TO SAM1;
  X = NORMAL(SEED) * 4;
  TOTX = TOTX + X;
  SSX = SSX + X * X;
  N+1;
*PUT X= TOTX= SSX= N=  SAM1=;
  END;
XBAR = TOTX / N;
VARX = (SSX - N * XBAR**2) / (N-1);
T = XBAR / SQRT(VARX/N);
DF = N - 1;
P_T = 1-PROBT(T,DF);
   IF P_T < = .05 THEN DO;
   CK + 1;
   ACTUAL = CK / ITERATE;
   END;
KEEP SEED SAMPLE ITERATE CK ACTUAL;
END;
CARDS;
5787747  10  100000  .20
;
DATA NORM_40;
INPUT SEED SAMPLE ITERATE PROP;
SAM1 = PROP * SAMPLE;
SAM2 = SAMPLE - SAM1;
```

```
DO I = 1 TO ITERATE;
TOTX = 0;
SSX = 0;
N = 0;
  DO J = 1 TO SAM2;
  X = NORMAL(SEED);
  TOTX = TOTX + X;
  SSX = SSX + X * X;
  N+1;
*PUT X= TOTX= SSX= N=  SAM2=;
  END;
  DO J = 1 TO SAM1;
  X = NORMAL(SEED) * 4;
  TOTX = TOTX + X;
  SSX = SSX + X * X;
  N+1;
*PUT X= TOTX= SSX= N=  SAM1=;
  END;
XBAR = TOTX / N;
VARX = (SSX - N * XBAR**2) / (N-1);
T = XBAR / SQRT(VARX/N);
DF = N - 1;
P_T = 1-PROBT(T,DF);
   IF P_T < = .05 THEN DO;
   CK + 1;
   ACTUAL = CK / ITERATE;
   END;
KEEP SEED SAMPLE ITERATE CK ACTUAL;
END;
CARDS;
1272301  10  100000  .40
;
DATA ALL;
 SET NORMAL NORM_10 NORM_20 NORM_40;
PROC PRINT;
DATA DIFF;
ARRAY DIFD(4) V1-V4;
DO I = 1 TO 4;
 SET ALL;
DIFD(I) = ACTUAL;
END;
DIF_12 = V1 - V2;
DIF_23 = V2 - V3;
DIF_34 = V3 - V4;
KEEP DIF_12 DIF_23 DIF_34;
PROC PRINT;
RUN;
```

OBS	SEED	SAMPLE	ITERATE	CK	ACTUAL
1	3445167	10	1000	46	0.046
2	9221023	10	1000	44	0.044
3	5787747	10	1000	49	0.049
4	1272301	10	1000	37	0.037

OBS	DIF_12	DIF_23	DIF_34
1	.002	-.005	0.012

OBS	SEED	SAMPLE	ITERATE	CK	ACTUAL
1	3445167	10	10000	461	0.0461
2	9221023	10	10000	405	0.0405
3	5787747	10	10000	440	0.0440
4	1272301	10	10000	450	0.0450

OBS	DIF_12	DIF_23	DIF_34
1	.0056	-.0035	-.001

OBS	SEED	SAMPLE	ITERATE	CK	ACTUAL
1	3445167	10	50000	2429	0.04858
2	9221023	10	50000	2007	0.04014
3	5787747	10	50000	1937	0.03874
4	1272301	10	50000	2220	0.04440

OBS	DIF_12	DIF_23	DIF_34
1	.00844	.0014	-.00566

OBS	SEED	SAMPLE	ITERATE	CK	ACTUAL
1	3445167	10	100000	4924	0.04924
2	9221023	10	100000	4022	0.04022
3	5787747	10	100000	3894	0.03894
4	1272301	10	100000	4420	0.04420

OBS	DIF_12	DIF_23	DIF_34
1	.00902	.00128	-.00526

Appendix B
SAS Programming Code Illustrating
Olson's Method: Generate A Single Set Of Data
And Perform All Treatments On These Data
And Use Monte Carlo Critical Values.

```
DATA NORMAL;
INPUT SEED SAMPLE ITERATE;
DO I = 1 TO ITERATE;
TOTX = 0;
SSX = 0;
N = 0;
  DO J = 1 TO SAMPLE;
  X = NORMAL(SEED);
  TOTX = TOTX + X;
  SSX = SSX + X * X;
  N+1;
  END;
XBAR = TOTX / N;
VARX = (SSX - N * XBAR**2) / (N-1);
T = XBAR / SQRT(VARX/N);
   IF T > = 1.77145 THEN DO;
   CK + 1;
   ACTUAL = CK / ITERATE;
   END;
KEEP SEED SAMPLE ITERATE CK ACTUAL;
END;
CARDS;
3445167  10  1000
;
*;
DATA NORM_20;
INPUT SEED SAMPLE ITERATE PROP;
SAM1 = PROP * SAMPLE;
SAM2 = SAMPLE - SAM1;
DO I = 1 TO ITERATE;
TOTX = 0;
SSX = 0;
N = 0;
  DO J = 1 TO SAM2;
  X = NORMAL(SEED);
  TOTX = TOTX + X;
  SSX = SSX + X * X;
  N+1;
 PUT X= TOTX= SSX= N=  SAM2=;
  END;
  DO J = 1 TO SAM1;
  X = NORMAL(SEED) * 4;
  TOTX = TOTX + X;
  SSX = SSX + X * X;
  N+1;
 PUT X= TOTX= SSX= N=  SAM1=;
```

```
  END;
XBAR = TOTX / N;
VARX = (SSX - N * XBAR**2) / (N-1);
T = XBAR / SQRT(VARX/N);
   IF T > = 1.77145 THEN DO;
   CK + 1;
   ACTUAL = CK / ITERATE;
   END;
KEEP SEED SAMPLE ITERATE CK ACTUAL;
END;
CARDS;
3445167  10  1000  .20
;
*;
DATA NORM_40;
INPUT SEED SAMPLE ITERATE PROP;
SAM1 = PROP * SAMPLE;
SAM2 = SAMPLE - SAM1;
DO I = 1 TO ITERATE;
TOTX = 0;
SSX = 0;
N = 0;
  DO J = 1 TO SAM2;
  X = NORMAL(SEED);
  TOTX = TOTX + X;
  SSX = SSX + X * X;
  N+1;
PUT X= TOTX= SSX= N=  SAM2=;
  END;
  DO J = 1 TO SAM1;
  X = NORMAL(SEED) * 4;
  TOTX = TOTX + X;
  SSX = SSX + X * X;
  N+1;
PUT X= TOTX= SSX= N=  SAM1=;
  END;
XBAR = TOTX / N;
VARX = (SSX - N * XBAR**2) / (N-1);
T = XBAR / SQRT(VARX/N);
   IF T > = 1.77145 THEN DO;
   CK + 1;
   ACTUAL = CK / ITERATE;
   END;
KEEP SEED SAMPLE ITERATE CK ACTUAL;
END;
CARDS;
3445167  10  1000  .40
;
*;
DATA NORM_60;
INPUT SEED SAMPLE ITERATE PROP;
SAM1 = PROP * SAMPLE;
SAM2 = SAMPLE - SAM1;
DO I = 1 TO ITERATE;
```

```
TOTX = 0;
SSX = 0;
N = 0;
  DO J = 1 TO SAM2;
  X = NORMAL(SEED);
  TOTX = TOTX + X;
  SSX = SSX + X * X;
  N+1;
PUT X= TOTX= SSX= N=  SAM2=;
  END;
  DO J = 1 TO SAM1;
  X = NORMAL(SEED) * 4;
  TOTX = TOTX + X;
  SSX = SSX + X * X;
  N+1;
PUT X= TOTX= SSX= N=  SAM1=;
  END;
XBAR = TOTX / N;
VARX = (SSX - N * XBAR**2) / (N-1);
T = XBAR / SQRT(VARX/N);
   IF T > = 1.77145 THEN DO;
   CK + 1;
   ACTUAL = CK / ITERATE;
   END;
KEEP SEED SAMPLE ITERATE CK ACTUAL;
END;
CARDS;
3445167  10  1000  .60
;
*;
DATA ALL;
 SET NORMAL NORM_20 NORM_40 NORM_60;
PROC PRINT;
DATA DIFF;
ARRAY DIF(4) V1-V4;
DO I = 1 TO 4;
 SET ALL;
DIF(I) = ACTUAL;
END;
DIF_12 = V1 - V2;
DIF_23 = V2 - V3;
DIF_34 = V3 - V4;
KEEP DIF_12 DIF_23 DIF_34;
PROC PRINT;
RUN;
```

OBS	SEED	SAMPLE	ITERATE	CK	ACTUAL
1	3445167	10	1000	50	0.050
2	3445167	10	1000	42	0.042
3	3445167	10	1000	39	0.039
4	3445167	10	1000	47	0.047

OBS	DIF_12	DIF_23	DIF_34
1	.008	.003	-.008

OBS	SEED	SAMPLE	ITERATE	CK	ACTUAL
1	3445167	10	10000	500	0.0500
2	3445167	10	10000	406	0.0406
3	3445167	10	10000	437	0.0437
4	3445167	10	10000	473	0.0473

OBS	DIF_12	DIF_23	DIF_34
1	.0094	-.0031	-.0036

OBS	SEED	SAMPLE	ITERATE	CK	ACTUAL
1	3445167	10	50000	2500	0.05000
2	3445167	10	50000	2045	0.04090
3	3445167	10	50000	2017	0.04034
4	3445167	10	50000	2341	0.04682

OBS	DIF_12	DIF_23	DIF_34
1	.0091	.00056	-.00648

OBS	SEED	SAMPLE	ITERATE	CK	ACTUAL
1	3445167	10	100000	5000	0.05000
2	3445167	.10	100000	4065	0.04065
3	3445167	10	100000	3957	0.03957
4	3445167	10	100000	4576	0.04576

OBS	DIF_12	DIF_23	DIF_34
1	.00935	.00108	-.00619

BEST COPY AVAILABLE

Appendix C
Output Using The Modified Olson Method

OBS	SEED	SAMPLE	ITERATE	CK	ACTUAL
1	3445167	10	1000	46	0.046
2	3445167	10	1000	35	0.035
3	3445167	10	1000	35	0.035
4	3445167	10	1000	44	0.044

OBS	DIF_12	DIF_23	DIF_34
1	0.011	0	-.009

OBS	SEED	SAMPLE	ITERATE	CK	ACTUAL
1	3445167	10	10000	461	0.0461
2	3445167	10	10000	357	0.0357
3	3445167	10	10000	373	0.0373
4	3445167	10	10000	419	0.0419

OBS	DIF_12	DIF_23	DIF_34
1	0.0104	-.0016	-.0046

OBS	SEED	SAMPLE	ITERATE	CK	ACTUAL
1	3445167	10	50000	2429	0.04858
2	3445167	10	50000	1965	0.03930
3	3445167	10	50000	1916	0.03832
4	3445167	10	50000	2253	0.04506

OBS	DIF_12	DIF_23	DIF_34
1	.00928	.00098	-.00674

OBS	SEED	SAMPLE	ITERATE	CK	ACTUAL
1	3445167	10	100000	4924	0.04924
2	3445167	10	100000	3977	0.03977
3	3445167	10	100000	3875	0.03875
4	3445167	10	100000	4487	0.04487

OBS	DIF_12	DIF_23	DIF_34
1	.00947	.00102	-.00612

Appendix D
Sawilowsky and Blair's Fortran Code
For Generating Data From A Population
Like Those Found In Real Life
(Here Only The Multimodal Lumpy Achievement Measures)

```
C Multimodal Lumpy, Achievement
C MODIFIED FOR THIS EXAMPLE BY BARCIKOWSKI, MARCH 1999
C ************************************************************
      DIMENSION IR(467), AIR(467), TED(467)
      OPEN(4,FILE='SAW.TXT')
      OPEN(5)
      OPEN(6,FILE='SAW.OUT')
C NRA  IS THE SAMPLE SIZE
C IDUM IS THE NEGATIVE RANDOM NUMBER SEED
C KPOP IS THE SIZE OF THE POPULATION
C IR   IS A VECTOR CONTAINING THE ELEMENTS TO BE
C      SAMPLED FROM THE POPULATION
C************************************************************
      READ(4,*) NRA, IDUM, KPOP, ITERATE
      PRINT 5, NRA, IDUM, KPOP, ITERATE
    5 FORMAT (1H0,'THE SAMPLE SIZE IS:',I7,/,
     <1H0,'THE SEED IS:',I10,/,1H0,'THE POPULATION SIZE IS:',I7,
     <1H0,'THE NUMBER OF ITERATIONS IS:',I10)
C ************************************************************
C     READ IN TED'S (MICCERI, 1987, FERA) TABLE 8:
C     ACHIEVEMENT MEASURE EXHIBITING MULTIMODALITY
C     AND LUMPINESS.  K=467
C ************************************************************
      K=467
      DO 710 I=1,5
        TED(I)=0.
  710 CONTINUE
      DO 711 I=1,8
        TED(5+I)=1.
  711 CONTINUE
      DO 712 I=1,8
        TED(13+I)=2.
  712 CONTINUE
      DO 713 I=1,3
        TED(21+I)=3.
  713 CONTINUE
      DO 714 I=1,8
        TED(24+I)=4.
  714 CONTINUE
      DO 715 I=1,6
        TED(32+I)=5.
  715 CONTINUE
      DO 716 I=1,3
        TED(38+I)=6.
  716 CONTINUE
      DO 717 I=1,9
        TED(41+I)=7.
  717 CONTINUE
      DO 718 I=1,12
        TED(50+I)=8.
  718 CONTINUE
      DO 719 I=1,18
        TED(62+I)=9.
  719 CONTINUE
      DO 720 I=1,11
        TED(80+I)=10.
  720 CONTINUE
      DO 721 I=1,23
```

BEST COPY AVAILABLE

```
      TED(91+I)=11.
721 CONTINUE
    DO 722 I=1,22
      TED(114+I,=12.
722 CONTINUE
    DO 723 I=1,24
      TED(136+I)=13.
723 CONTINUE
    DO 724 I=1,20
      TED(160+I)=14.
724 CONTINUE
    DO 725 I=1,15
      TED(180+I)=15.
725 CONTINUE
    DO 726 I=1,18
      TED(195+I)=16.
726 CONTINUE
    DO 727 I=1,12
      TED(213+I)=17.
727 CONTINUE
    DO 728 I=1,9
      TED(225+I)=18.
728 CONTINUE
    DO 729 I=1,10
      TED(234+I)=19.
729 CONTINUE
    DO 730 I=1,10
      TED(244+I)=20.
730 CONTINUE
    DO 731 I=1,7
      TED(254+I)=21.
731 CONTINUE
    DO 732 I=1,8
      TED(261+I)=22.
732 CONTINUE
    DO 733 I=1,10
      TED(269+I)=23.
733 CONTINUE
    DO 734 I=1,3
      TED(279+I)=24.
734 CONTINUE
    DO 735 I=1,5
      TED(282+I)=25.
735 CONTINUE
    DO 736 I=1,10
      TED(287+I)=26.
736 CONTINUE
    DO 737 I=1,9
      TED(297+I)=27.
737 CONTINUE
    DO 738 I=1,?
      TED(306+I)=28.
738 CONTINUE
    DO 739 I=1,10
      TED(309+I)=29.
739 CONTINUE
    DO 740 I=1,6
      TED(319+I)=30.
740 CONTINUE
    DO 741 I=1,11
      TED(325+I)=31.
741 CONTINUE
    DO 742 I=1,15
      TED(336+I)=32.
742 CONTINUE
    DO 743 I=1,13
      TED(351+I)=33.
743 CONTINUE
```

BEST COPY AVAILABLE

```
      DO 744 I=1,15
        TED(364+I)=34.
  744 CONTINUE
      DO 745 I=1,10
        TED(379+I)=35.
  745 CONTINUE
      DO 746 I=1,12
        TED(389+I)=36.
  746 CONTINUE
      DO 747 I=1,17
        TED(401+I)=37.
  747 CONTINUE
      DO 748 I=1,10
        TED(418+I)=38.
  748 CONTINUE
      DO 749 I=1,6
        TED(428+I)=39.
  749 CONTINUE
      DO 750 I=1,11
        TED(434+I)=40.
  750 CONTINUE
      DO 751 I=1,9
        TED(445+I)=41.
  751 CONTINUE
      DO 752 I=1,6
        TED(454+I)=42.
  752 CONTINUE
      DO 753 I=1,7
        TED(460+I)=43.
  753 CONTINUE
      RN = NRA
      D1 = 1
      D2 = RN - 1.0
      ALPHA = .05
      DO 14 IBIG = 1, ITERATE
      TOTX = 0.0
      SSX = 0.0
      DO 1 I = 1, KPOP
      AIR(I) = RAN1(IDUM)
      IR(I) = I
    1 CONTINUE
      CALL SORT(IR,AIR,KPOP)
      DO 2 I = 1, NRA
    2 CONTINUE
      DO 3 J = 1, NRA
      II = IR(J)
      X = TED(II)
      TOTX = TOTX + X
      SSX = SSX + X * X
    3 CONTINUE
      XBAR = TOTX / RN
      VARX = (SSX -RN * XBAR**2) / (RN-1)
C THE MEAN OF THE POPULATION IS 21.1477516; HA: u > 21.1477516
      T = (XBAR-21.1477516) / SQRT(VARX/RN)
      IF (T .LE. 0.0) GO TO 14
      F = T * T
      P = BETAI(0.5*D2, 0.5*D1, D2/(D2+D1*F))
      P = P/2.0
      IF (P .LE. .05) THEN
      CK = CK + 1
      ENDIF
   14 CONTINUE
      ACTUAL = CK / ITERATE
      PRINT 15, CK, ACTUAL
   15 FORMAT(1X,'THE NUMBER OF REJECTIONS:',F7.2,/,
     <1X,'THE ACTUAL ALPHA:',F8.4)
      STOP
      END
```

BEST COPY AVAILABLE

```
      FUNCTION RAN1(IDUM)
      DIMENSION R(97)
      PARAMETER (M1=259200,IA1=7141,IC1=54773,RM1=3.8580247E-6)
      PARAMETER (M2=134456,IA2=8121,IC2=28411,RM2=7.4373773E-6)
      PARAMETER (M3=243000,IA3=4561,IC3=51349)
      DATA IFF /0/
      IF (IDUM.LT.0.OR.IFF.EQ.0) THEN
        IFF=1
        IX1=MOD(IC1-IDUM,M1)
        IX1=MOD(IA1*IX1+IC1,M1)
        IX2=MOD(IX1,M2)
        IX1=MOD(IA1*IX1+IC1,M1)
        IX3=MOD(IX1,M3)
        DO 11 J=1,97
          IX1=MOD(IA1*IX1+IC1,M1)
          IX2=MOD(IA2*IX2+IC2,M2)
          R(J)=(FLOAT(IX1)+FLOAT(IX2)*RM2)*RM1
11      CONTINUE
        IDUM=1
      ENDIF
      IX1=MOD(IA1*IX1+IC1,M1)
      IX2=MOD(IA2*IX2+IC2,M2)
      IX3=MOD(IA3*IX3+IC3,M3)
      J=1+(97*IX3)/M3
      IF(J.GT.97.OR.J.LT.1)PAUSE
      RAN1=R(J)
      R(J)=(FLOAT(IX1)+FLOAT(IX2)*RM2)*RM1
      RETURN
      END
C                                                                      RMA01680
C*********************************************************             RMA01690
C  SORT THE UNIFORM NOS LARGEST TO SMALLEST
C*********************************************************             RMA01710
C                                                                      RMA01720
      SUBROUTINE SORT(NUMS,RIKE, NOCELL)
      DIMENSION RIKE(467), NUMS(467)
   20 FLAG = 0                                                         RMA01760
      DO 10 I = 1, NOCELL-1                                            RMA01770
      IF(RIKE(I) .LT. RIKE(I+1)) THEN
         TEMP = RIKE(I)
         RIKE(I) = RIKE(I+1)
         RIKE(I+1) = TEMP
         LT = NUMS(I)
         NUMS(I) = NUMS(I+1)
         NUMS(I+1) = LT
         FLAG = 1                                                      RMA01850
       END IF                                                          RMA01860
   10 CONTINUE                                                         RMA01870
      IF (FLAG .EQ. 1) THEN                                            RMA01880
       GO TO 20                                                        RMA01890
      END IF                                                           RMA01900
      RETURN                                                           RMA01910
      END                                                              RMA01920
      FUNCTION BETAI(A,B,X)
      IF(X.LT.0..OR.X.GT.1.)PAUSE 'bad argument X in BETAI'
      IF(X.EQ.0..OR.X.EQ.1.)THEN
        BT=0.
      ELSE
        BT=EXP(GAMMLN(A+B)-GAMMLN(A)-GAMMLN(B)
     *      +A*ALOG(X)+B*ALOG(1.-X))
      ENDIF
      IF(X.LT.(A+1.)/(A+B+2.))THEN
        BETAI=BT*BETACF(A,B,X)/A
        RETURN
      ELSE
        BETAI=1.-BT*BETACF(B,A,1.-X)/B
        RETURN
      ENDIF
```

```
      END
      FUNCTION BETACF(A,B,X)
      PARAMETER (ITMAX=100,EPS=3.E-7)
      AM=1.
      BM=1.
      AZ=1.
      QAB=A+B
      QAP=A+1.
      QAM=A-1.
      BZ=1.-QAB*X/QAP
      DO 11 M=1,ITMAX
        EM=M
        TEM=EM+EM
        D=EM*(B-M)*X/((QAM+TEM)*(A+TEM))
        AP=AZ+D*AM
        BP=BZ+D*BM
        D=-(A+EM)*(QAB+EM)*X/((A+TEM)*(QAP+TEM))
        APP=AP+D*AZ
        BPP=BP+D*BZ
        AOLD=AZ
        AM=AP/BPP
        BM=BP/BPP
        AZ=APP/BPP
        BZ=1.
        IF(ABS(AZ-AOLD).LT.EPS*ABS(AZ))GO TO 1
11    CONTINUE
      PAUSE 'A or B too big, or ITMAX too small'
1     BETACF=AZ
      RETURN
      END
      FUNCTION GAMMLN(XX)
      REAL*8 COF(6),STP,HALF,ONE,FPF,X,TMP,SER
      DATA COF,STP/76.18009173D0,-86.50532033D0,24.01409822D0,
     *    -1.231739516D0,.120858003D-2,-.536382D-5,2.50662827465D0/
      DATA HALF,ONE,FPF/0.5D0,1.0D0,5.5D0/
      X=XX-ONE
      TMP=X+FPF
      TMP=(X+HALF)*LOG(TMP)-TMP
      SER=ONE
      DO 11 J=1,6
        X=X+ONE
        SER=SER+COF(J)/X
11    CONTINUE
      GAMMLN=TMP+LOG(STP*SER)
      RETURN
      END
```

OUTPUT

```
THE SAMPLE SIZE IS:      10

THE SEED IS: -35612893

THE POPULATION SIZE IS:    467

THE NUMBER OF ITERATIONS IS:       1000
THE NUMBER OF REJECTIONS:  37.00
THE ACTUAL ALPHA:  0.0370
```